For the customer’s mobile application, I developed unit tests for the contact, task, and appointment services. The software testing techniques that were employed in the project were based on the specific project at hand. However, some common techniques included specific testing such as: unit testing, integration testing, functional testing, and acceptance testing. These techniques were used to verify that the software works as expected, meets requirements, and performs flawlessly. To ensure the tests were effective, I used JUnit test cases to test for edge cases and to examine the code for errors. In doing so, I was able to cover 95.6% of the code, verifying that it was effective and efficient. Other software testing techniques that could have been learned about include performance testing, security testing, and usability testing. These techniques focus on verifying specific aspects of the software, such as its performance, security, and usability.

My personal experience writing the JUnit tests were positive. I was able to ensure that my code was technically sound by verifying that the logic was correct and that the expected outputs were produced. To ensure the code was efficient, I used a combination of methods, such as refactoring and removing redundant code. I was also able to use JUnit assertions to ensure that the expected results were constructed.

The practical uses and implications for different software development projects and situations were also based on the specific project. For example, unit testing is often used in agile development methodologies to ensure that each piece of code is running properly before it is integrated with the rest of the system. Performance testing is often used in large systems to verify that the software will perform flawlessly under high-stress conditions.

The software testing technique that I used in this project was JUnit test cases to test for edge cases and to examine the code for errors. Properly managing software testing is also important because it ensures that the software is of high quality and meets the requirements of the stakeholders. The risks of not appropriately verifying and validating code include the potential for bugs and security vulnerabilities, which can lead to costly rework and loss of reputation.

The software testing techniques that I did not use for this project includes, performance and security testing. The major components of governing testing that should be examined and implemented includes testing strategy, planning, and monitoring. This would involve defining the scope of the testing effort, creating a plan for executing the tests, and monitoring the results to ensure that the software meets the desired criteria. Performance testing is used to measure the speed, scalability, and responsiveness of a system, while security testing is used to identify and mitigate potential security risks.

The practical use of the software testing technique I used is that Junit can be used to verify the correctness and efficiency of the code. Tools and technology available for implementing testing strategies include test management tools, automation tools, and performance testing tools. Automation could be used to support quality software development by reducing manual testing efforts and increasing testing coverage. However, it is important to use automation appropriately and not 100% rely solely on automation to ensure software quality. Automation can also be used to uncover errors and find edge cases. The insinuation of not using certain software testing techniques are that potential errors and security risks may go unnoticed.

When it comes to the mindset that I adopted while working on this project, I was highly cautious and aware of the complexity of the code. The mindset adopted when working as a software tester is one of caution and a focus on ensuring the best quality of the software. It is extremely important to appreciate the complexity and interrelationships of the code being tested to recognize and address any potential problems. I attempted to limit bias in my review of the code by using code reviews and unit tests to thoroughly examine the functionality of the code. To limit bias in code review, it is crucial to approach the code objectively and follow a systematic testing process. Bias can become a concern when testing one's own code, as the developer may not be as critical in their review as an independent tester would be. As a software developer, it is crucial to be aware of the potential for bias and to be diligent in testing one’s own code.

Finally, I believe it is important to be disciplined in the commitment to quality when writing and testing code. Being disciplined in the commitment made to quality as a software engineering professional is very crucial to produce high-quality software that meets the needs of the stakeholders. Cutting corners when writing or testing code can lead to technical debt, which can also result in costly rework and loss of reputation. To avoid technical debt, it is very important to follow best practices and consistently strive for quality in the work that is done.